|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-3 «Управление и моделирование систем»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № \_\_\_**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
*(фио студента)*

Проверил:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2020 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью указателей (однонаправленных/двунаправленный динамический линенйый связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктур, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № 37.**

**Реализация связи элементов линейного списка: Очередь**

**Способ организации линейного связанный список: Массив**

**Алгоритм сортировки: Хоара с медианным элементом**

**Теория о сортировках.**

**Сортировка естественным двухпутевым слиянием.**

МБыстрая сортировка, сортировка Хоара ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) quicksort), часто называемая qsort (по имени в [стандартной библиотеке языка Си](https://ru.wikipedia.org/wiki/Стандартная_библиотека_языка_Си)) — [алгоритм сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки), разработанный английским информатиком [Чарльзом Хоаром](https://ru.wikipedia.org/wiki/Хоар,_Чарльз_Энтони_Ричард) во время его работы в [МГУ](https://ru.wikipedia.org/wiki/МГУ) в [1960 году](https://ru.wikipedia.org/wiki/1960_год).

## **Общее описание**

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «[Пузырьковая сортировка](https://ru.wikipedia.org/wiki/Пузырьковая_сортировка)» и «[Шейкерная сортировка](https://ru.wikipedia.org/wiki/Шейкерная_сортировка)»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. (Таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.)

Общая идея алгоритма состоит в следующем:

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие»[[1]](https://ru.wikipedia.org/wiki/Быстрая_сортировка" \l "cite_note-1).
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения

**Листинг программы с расчетами.**

**ФАЙЛ main.c**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "queue.h"

void timespec\_diff(struct timespec \*a, struct timespec \*b, struct timespec \*result) {

result->tv\_sec = a->tv\_sec - b->tv\_sec;

result->tv\_nsec = a->tv\_nsec - b->tv\_nsec;

if (result->tv\_nsec < 0) {

--result->tv\_sec;

result->tv\_nsec += 1000000000L;

}

}

int main()

{

// clock\_gettime()

srand(time(NULL));

unsigned long long result=0;

unsigned long long trash=0;

struct timespec before,after,result\_t;

for(int n=1;n<3002;n+=300){

result=0;

trash=0;

struct queue \*q=queue\_new(&trash);

for(int i=0;i<n;i++){

queue\_add(q,rand()%1000,&trash);

}

clock\_gettime(CLOCK\_REALTIME,&before);

queue\_sort(q,0,q->length-1,&result);

clock\_gettime(CLOCK\_REALTIME,&after);

queue\_delete(q,&trash);

timespec\_diff(&after,&before,&result\_t);

printf("%d %llu ",n,result);

printf("%lld.%.9ld\n", (long long)result\_t.tv\_sec, result\_t.tv\_nsec);

fflush(stdout);

}

return 0;

}

**ФАЙЛ queue.h**

#ifndef QUEUE\_H

#define QUEUE\_H

struct queue{

unsigned int length;

int \*aptr;

int \*last;

};

void \*queue\_new(unsigned long long \*return\_n);

char queue\_isempty(struct queue \*q, unsigned long long \*return\_n);

void queue\_add(struct queue \*q, int value, unsigned long long \*return\_n);

int queue\_pop(struct queue \*q, unsigned long long \*return\_n);

struct queue \*queue\_copy(struct queue \*q, unsigned long long \*return\_n);

void queue\_delete(struct queue \*q, unsigned long long \*return\_n);

void queue\_set(struct queue \*q, unsigned int index, int value, unsigned long long \*return\_n);

void queue\_sort(struct queue \*q, int left, int right, unsigned long long \*return\_n);

void queue\_out(struct queue \*q);

#endif // QUEUE\_H

**ФАЙЛ queue.c**

#include "queue.h"

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

int sort\_called=0;

int part\_called=0;

//Итого 10

void \*queue\_new(unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

struct queue \*q=malloc(sizeof(struct queue));

n+=3;

q->length=0;

n+=2;

q->aptr=NULL;

n+=2;

q->last=NULL;

n+=2;

n+=1;

\*return\_n=n;

return q;

}

//Итого 3

char queue\_isempty(struct queue \*q, unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

n+=3;

\*return\_n=n;

return ((q->length)==0);//1+1+1

}

//Итого 29

void queue\_add(struct queue \*q, int value, unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

if(q==NULL)return;

if((q->aptr==NULL)&&(q->length!=0)){//1+1+1+1+1

n+=5;

q->length=1;

n+=2;

q->aptr=malloc(sizeof(int));

n+=3;

q->aptr[0]=value;

n+=3;

q->last=q->aptr+(q->length-1);

n+=6;

\*return\_n=n;

return;

}else{

q->aptr=realloc(q->aptr,(q->length+1)\*sizeof(int));//1+1+1+1+1+1+1+1+1

n+=9;

memmove(q->aptr+1,q->aptr,(q->length)\*sizeof(int));//1+1+1+1+1+1+1+1+1

n+=9;

q->aptr[0]=value;//1+1+1

n+=3;

q->length+=1;//1+1

n+=2;

q->last=q->aptr+(q->length-1);//1+1+1+1+1+1

n+=6;

\*return\_n=n;

}

}

//Итого 21

int queue\_pop(struct queue \*q, unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

if(q==NULL)return 0;

if(q->length==0)return 0;

int result=\*(q->last);//1+1+1

n+=3;

q->aptr=realloc(q->aptr,(q->length-1)\*sizeof(int));//1+1+1+1+1+1+1+1+1

n+=9;

q->length-=1;//1+1

n+=2;

q->last=q->aptr+(q->length-1);//1+1+1+1+1+1

n+=6;

n+=1;

\*return\_n=n;

return result;//+1

}

//В данной функции оба цикла в сумме срабатывают n раз, и так как их тела одинаковы, примем их за один цикл для I от 0 до n

//Итого 61+63n

void queue\_set(struct queue \*q, unsigned int index, int value, unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

n+=1;

for(int i=index;i<q->length-1;i++){

n+=4;

queue\_add(q,queue\_pop(q,&n),&n);//29+21

n+=9;

}

queue\_pop(q,&n);//21

n+=4;

queue\_add(q,value,&n);//29

n+=5;

n+=1;

for(int i=0;i<index;i++){

n+=2;

queue\_add(q,queue\_pop(q,&n),&n);//29+21

n+=9;

}

\*return\_n=n;

}

//Тело функции крайне схоже с телом функции выше

//Итого 62+63n

int queue\_get(struct queue \*q, unsigned int index, unsigned long long \*return\_n){

unsigned long long n=\*return\_n;

n+=1;

for(int i=index;i<q->length-1;i++){

n+=4;

queue\_add(q,queue\_pop(q,&n),&n);

n+=9;

}

int result=queue\_pop(q,&n);

n+=4;

queue\_add(q,result,&n);

n+=5;

n+=1;

for(int i=0;i<index;i++){

n+=2;

queue\_add(q,queue\_pop(q,&n),&n);

n+=9;

}

n+=1;

\*return\_n=n;

return result;

}

//Данную функцию можно не учитывать

void queue\_out(struct queue \*q){

unsigned long long trash=0;

if(q==NULL) return;

for(int i=0;i<q->length;i++)

printf("%d ",q->aptr[i]);

printf("\n");

}

//Мною было экспериментально установлено, что количество вызовов этой фунции равно f(n)=0.67\*n

//Итого 267\*ln(n)\*n+ln(n)\*252\*n^2+229\*n

int partition (struct queue \*q, int low, int high, unsigned long long \*return\_n) //0.67\*n

{

unsigned long long n=\*return\_n;

int pivot = queue\_get(q,high,&n);

n+=5;

int i = (low - 1);

n+=1;

n+=1;

for (int j = low; j <= high- 1; j++)

{

n+=3;

n+=6;

if (queue\_get(q,j,&n) <= pivot)

{

//Т.к. данный блок кода находится в цикле, количество входов в него стоило бы принять за n, но т.к. цикл идет не от 0 до n, и на него накладывается условие, которое не всегда исполняется, примем количество входов в данный блок за полтора натуральных логарифмов от n.

i++;

n+=1;

int t=queue\_get(q,i,&n);

n+=5;

queue\_set(q,i,queue\_get(q,j,&n),&n);

n+=11;

queue\_set(q,j,t,&n);

n+=6;

}

}

int t=queue\_get(q,i+1,&n);

n+=6;

queue\_set(q,i+1,queue\_get(q,high,&n),&n);

n+=11;

queue\_set(q,high,t,&n);

n+=6;

n+=2;

\*return\_n=n;

return (i + 1);

}

//Мною было экспериментально установлено, что количество вызовов этой фунции равно f(n)=1.34\*n

//Итого 16 без учета частоты вызова и сложностей функций, вызываемых из данной

void queue\_sort(struct queue \*q, int low, int high, unsigned long long \*return\_n) //1.34\*n

{

unsigned long long n=\*return\_n;

n+=1;

if (low < high)

{

int pivot = partition(q, low, high,&n); //Сложности тел фунцкций можно игнорировать, т.к. мы знаем частоту их вызовов

n+=5;

queue\_sort(q, low, pivot - 1,&n);

n+=5;

queue\_sort(q, pivot + 1, high,&n);

n+=5;

}

\*return\_n=n;

}

// Итого 7

void queue\_delete(struct queue \*q, unsigned long long \*return\_n){//7

unsigned long long n=\*return\_n;

if(q==NULL) return;

q->length=0;//1+1

n+=2;

free(q->aptr);//1+1+1

n+=3;

free(q);//1+1

n+=2;

\*return\_n=n;

}

С учетом частоты вызова получим

F(n)= 267\*ln(n)\*n+ln(n)\*252\*n^2+250\*n

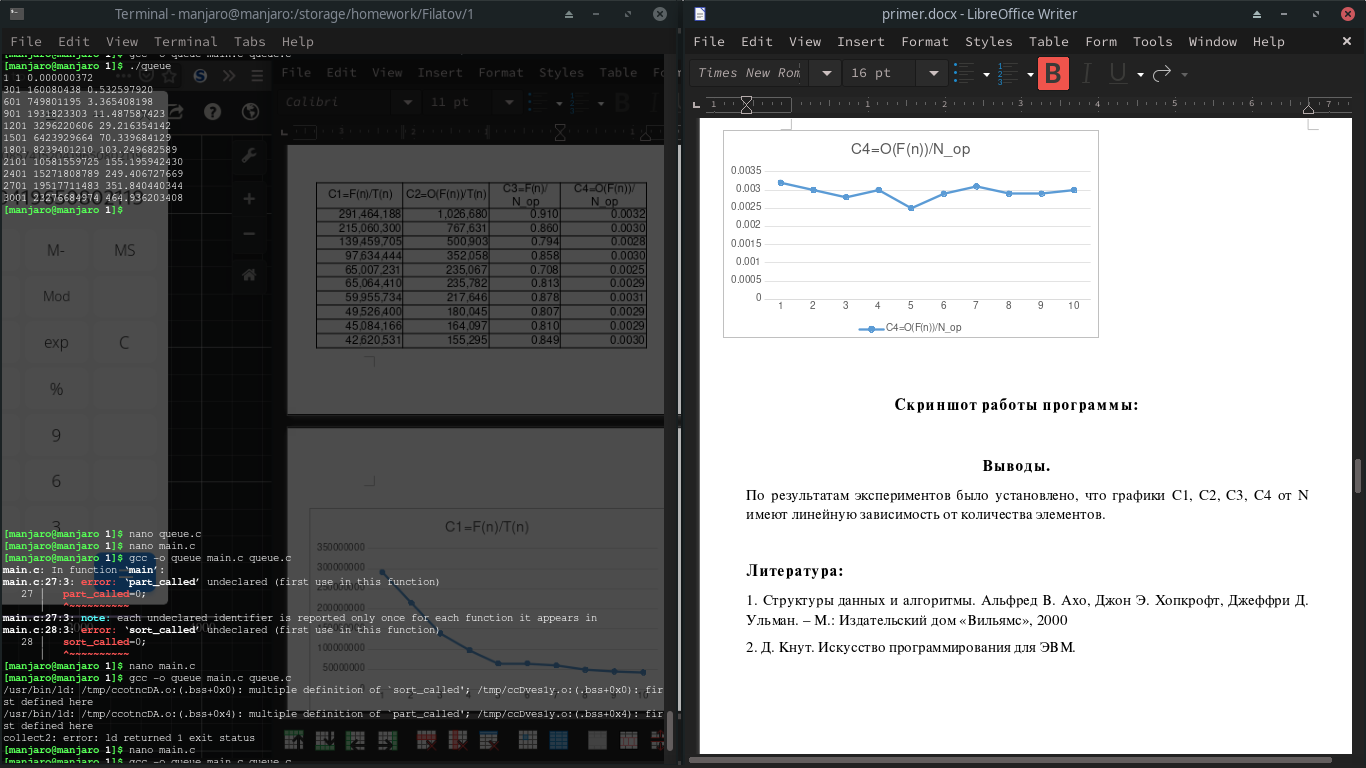
O(F(n))=n\*n\*ln(n)

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 145732094 | 513340 | 0.5325 | 160080438 |
| 600 | 645180901 | 2302894 | 3.3654 | 749801195 |
| 900 | 1534056757 | 5509939 | 11.4875 | 1931823303 |
| 1200 | 2831398904 | 10209710 | 29.2163 | 3296220606 |
| 1500 | 4550506238 | 16454745 | 70.3396 | 6423929664 |
| 1800 | 6701634308 | 24285555 | 103.2496 | 8239401210 |
| 2100 | 9293138911 | 33735144 | 155.1959 | 10581559725 |
| 2400 | 12332073614 | 44831370 | 249.4067 | 15271808789 |
| 2700 | 15824542538 | 57598341 | 351.8404 | 19517711483 |
| 3000 | 19775926438 | 72057308 | 464.9362 | 23276684974 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 291,464,188 | 1,026,680 | 0.910 | 0.0032 |
| 215,060,300 | 767,631 | 0.860 | 0.0030 |
| 139,459,705 | 500,903 | 0.794 | 0.0028 |
| 97,634,444 | 352,058 | 0.858 | 0.0030 |
| 65,007,231 | 235,067 | 0.708 | 0.0025 |
| 65,064,410 | 235,782 | 0.813 | 0.0029 |
| 59,955,734 | 217,646 | 0.878 | 0.0031 |
| 49,526,400 | 180,045 | 0.807 | 0.0029 |
| 45,084,166 | 164,097 | 0.810 | 0.0029 |
| 42,620,531 | 155,295 | 0.849 | 0.0030 |

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют линейную зависимость от количества элементов.

**Литература:**

1. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2000

2. Д. Кнут. Искусство программирования для ЭВМ.